Trabajo Práctico 10 - Pruebas de Integración

1- Objetivos de Aprendizaje

- Adquirir conocimientos sobre conceptos referidos a pruebas de integración (integration tests).
- Generar y ejecutar pruebas de integración utilizado frameworks de código abierto.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 5 (Libro Ingeniería de Software: Cap 8)

3- Consignas a desarrollar en el trabajo práctico:

Conceptos generales explicaciones de los mismos

Pruebas de integración

Una prueba de integración tiene como objetivo probar el comportamiento de un componente o la integración entre un conjunto de componentes. El término prueba funcional se usa a veces como sinónimo para prueba de integración. Las pruebas de integración comprueban que todo el sistema funciona según lo previsto, por lo que reducen la necesidad de pruebas manuales intensivas.

Este tipo de pruebas le permiten traducir sus historias de usuario en un conjunto de pruebas. La prueba se asemejaría a una interacción esperada del usuario con la aplicación.

Frameworks de pruebas

Existen una gran variedad de herramientas o frameworks disponibles para las pruebas de integración, tanto para componentes del backend como del frontend. Estas pueden ser comerciales, de código abierto o desarrolladas y utilizadas internamente por las compañías de software.

Para este trabajo práctico vamos a probar aplicaciones web y rest y para ello utilizaremos Codeceptjs como ejemplo.

Selenium

Selenium es una herramienta de prueba de software automatizada y de código abierto para probar aplicaciones web. Tiene capacidades para operar en diferentes navegadores

y sistemas operativos. Selenium es un conjunto de herramientas que ayuda a los testers a automatizar las aplicaciones basadas en la web de manera más eficiente.

Podemos codificar las pruebas directamente en un lenguaje de programación, por ejemplo javascript y ejecutarlas como parte del proceso de CI/CD.

Codeceptjs https://codecept.io/

Codeceptjs es un framework end to end para pruebas de integración y de aceptación de usuario, es muy simple de usar y abstrae al que escribe los tests de trabajar directamente con el driver de Selenium o algún otro driver.

4- Desarrollo:

1- Familiarizarse con CodeceptJs

- El objeto **I** y sus funcionalidades básicas: https://codecept.io/basics

2- Testeando la página de GitHub

- Instalar NodeJs v12 o superior: https://nodejs.org/en/download/
- En un directorio, por ejemplo **.\proyectos\ut** ejecutar:

```
npx create-codeceptjs .
```

- Si esta utilizando codeceptjs 3.0.0, hay que actualizar a uno superior, por ejemplo 3.0.1
- Cambiar en packages.json `"codeceptjs": "^3.0.0",` por `"codeceptjs": "^3.0.1",` y ejecutar `npm install`
- Ininicializar un nuevo proyecto CodeceptJS:

```
npx codeceptjs init
```

- Elegimos las opciones por defecto, ponemos **github** cuando se nos pregunte por el nombre del primer test:

```
D:\repos\ucc\ing-soft-3-2020\proyectos\ut>npx codeceptjs init

  Welcome to CodeceptJS initialization tool
  It will prepare and configure a test environment for you
```

```
Installing to D:\repos\ucc\ing-soft-3-2020\proyectos\ut
? Where are your tests located? ./*_test.js
? What helpers do you want to use? (Use arrow keys)
> Playwright
  WebDriver
  Puppeteer
  TestCafe
  Protractor
  Nightmare
  Appium
? Where should logs, screenshots, and reports to be stored? ./output
? Do you want localization for tests? (See https://codecept.io/translation/)
English (no localization)
Configure helpers...
? [Playwright] Base url of site to be tested http://localhost
? [Playwright] Show browser window Yes
? [Playwright] Browser in which testing will be performed. Possible options:
chromium, firefox or webkit chromium

Steps file created at ./steps_file.js
Config created at D:\repos\ucc\ing-soft-3-2020\proyectos\ut\codecept.conf.js
Directory for temporary output files created at './output'
Intellisense enabled in D:\repos\ucc\ing-soft-3-
2020\proyectos\ut\jsconfig.json
TypeScript Definitions provide autocompletion in Visual Studio Code and other
IDEs
Definitions were generated in steps.d.ts

 Almost ready... Next step:
Creating a new test...
-----------------------
? Feature which is being tested (ex: account, login, etc) github
? Filename of a test github_test.js

Test for github_test.js was created in D:\repos\ucc\ing-soft-3-
2020\proyectos\ut\github_test.js

--
CodeceptJS Installed! Enjoy supercharged testing! 
Find more information at https://codecept.io
```

- Editar el archivo generado:

```
Feature('My First Test');
Scenario('test something', (I) => {
```

```
});
```

- Escribir un escenario de prueba:

```
Scenario('test something', (I) => {
  I.amOnPage('https://github.com');
  I.see('GitHub');
});
```

- Finalmente correr el test:
  `npx codeceptjs run --steps`
- Agregamos otras validaciones

```
Scenario('test something', ({ I }) => {
    I.amOnPage('https://github.com');
    I.see('GitHub');
    I.see('Built for developers')
    I.scrollPageToBottom()
    I.seeElement("//li[contains(.,'© 2020 GitHub, Inc.')]")
});
```

- Para generar selectores fácilmente utilizamos plugins como (Firefox o Chrome)
  - TruePath https://addons.mozilla.org/en-US/firefox/addon/truepath/
  - ChroPath https://chrome.google.com/webstore/detail/chropath/ljngjbnaijcbncmcnjfhigebomdlkcjo

3- Testeando la aplicación spring-boot

- En un directorio, por ejemplo **.\proyectos\spring-boot-it** ejecutar:

```
npx create-codeceptjs .
```

- Instalar CodeceptJS con la librería webdriverio
  `npm install codeceptjs chai --save-dev`
- Inicializar CodeceptJS: `npx codeceptjs init`

- Responder las preguntas. Aceptar valores por defecto. Cuando pregunte por url colocar `http://localhost:8080` y y el nombre de los tests poner `spring-boot`
- Editar el archivo generado `spring-boot_tests.js`:

```
Feature('spring-boot');

const expect = require('chai').expect;
const {I} = inject();

Scenario('Verify a successful call', async () => {
        const res = await I.sendGetRequest('/');
        expect(res.status).to.eql(200);
});

Scenario('Verify return value', async () => {
        const res = await I.sendGetRequest('/');
        //console.log(res);
        expect(res.data.message).to.eql('Spring boot says hello from a Docker
container');
});
```

- Reemplazar la sección helpers de codecept.conf.js por:

```
        helpers: {
                REST: {
                        endpoint: "http://localhost:8080",
                        onRequest: () => {
                        }
                }
        }
```

- Levantar la aplicación spring-boot en otra consola (usando java o Docker):

```
cd ./proyectos/spring-boot
java -jar target/spring-boot-sample-actuator-2.0.2.jar
```

- Ejecutar los tests desde la carpeta `.\proyectos\spring-boot-it`

```
npx codeceptjs run --steps
```

- Analizar resultados

4- Habilitar reportes para utilizarlos en CICD

- Instalar el módulo para reporting

```
npm i mocha-junit-reporter mocha-multi --save
```

- Reemplazar la key mocha en el archivo codecept.conf.js por:

```
        mocha:  {
    "reporterOptions": {
      "codeceptjs-cli-reporter": {
        "stdout": "-",
        "options": {
          "steps": true,
        }
      },
      "mocha-junit-reporter": {
        "stdout": "./output/console.log",
        "options": {
          "mochaFile": "./output/result.xml"
        },
        "attachments": true //add screenshot for a failed test
                  }
                }
        }
```

- Ejecutar los tests nuevamente

```
npx codeceptjs run --steps --reporter mocha-multi
```

- La salida compatible con Jenkins esta en ./output/results.xml

5- Integrar la ejecución en Jenkins

- Utilizando la funcionalidad de Junit test en Jenkins colectar estos resultados de la ejecución después del deployment.